main

October 19, 2019

[1]:
```
# A class to get the sentences from the dataset

class SentenceGetter(object):

    def __init__(self, data):
        self.n_sent = 1
        self.data = data
        self.empty = False
        aggregate_function = lambda s : [(w, p, t)
                                         for w, p, t in zip(s["Word"].values.
 ↪tolist(),
                                                            s["POS"].values.
 ↪tolist(),
                                                            s["Tag"].values.
 ↪tolist())]
        self.grouped = self.data.groupby("Sentence #").apply(aggregate_function)
        self.sentences = [s for s in self.grouped]

    def getNext(self):
        try:
            s = self.grouped["Sentence: {}".format(self.n_sent)]
            self.n_sent += 1
            return s
        except:
            print("Exception")
            self.empty = True
            return None
```

[2]:
```
import pandas as pd
import numpy as np

# read the annotated dataset from kaggle
data = pd.read_csv("ner_dataset.csv", encoding="latin1")
data = data.fillna(method="ffill")
#data.tail(10)

l_words = list(set(data["Word"].values))
```

```
n_words = len(l_words)

l_tags = list(set(data["Tag"].values))
n_tags = len(l_tags)

sentence_getter = SentenceGetter(data)
l_sentences = sentence_getter.sentences

# Prepare the data
n_max_seq_size = 75
d_indexed_words = {word: index + 1 for index, word in enumerate(l_words)}
d_indexed_tags = {tag: index for index, tag in enumerate(l_tags)}
```

[3]:
```
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

X = [[d_indexed_words[word[0]] for word in sentence] for sentence in␣
 ↪l_sentences]
X = pad_sequences(maxlen=n_max_seq_size, sequences=X, padding="post",␣
 ↪value=n_words-1)

y = [[d_indexed_tags[word[2]] for word in sentence] for sentence in l_sentences]
y = pad_sequences(maxlen=n_max_seq_size, sequences=y, padding="post",␣
 ↪value=d_indexed_tags["O"])

# changing the y-labels to categorical for training purposes
y = [to_categorical(idx, num_classes=n_tags) for idx in y]

# Split in Training and Test sets
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.1)
```

```
Using TensorFlow backend.
```

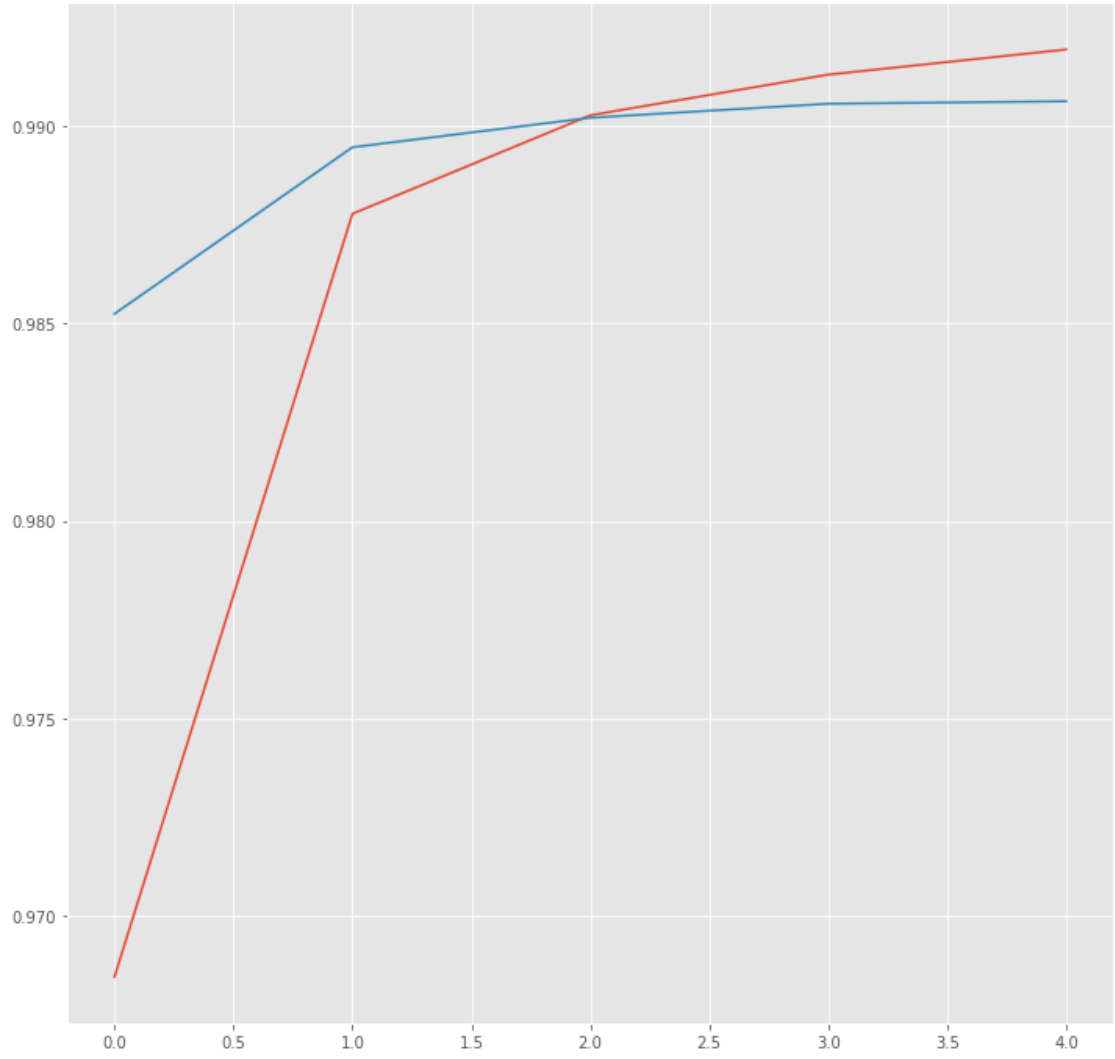
[6]:
```
# Now fitting a LSTM-CRF network with an embedding layer
from keras.models import Model, Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout,␣
 ↪Bidirectional
from keras_contrib.layers import CRF

model_input = Input(shape=(n_max_seq_size,))
model = Embedding(input_dim = n_words + 1,
                  output_dim = 20,
                  input_length = n_max_seq_size,
                  mask_zero = True)(model_input)  # 20-dim embedding

model = Bidirectional(LSTM(units=50,
```

```
                           return_sequences=True,
                           recurrent_dropout=0.1))(model)     # variational␣
 ↪biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model)  # a dense layer␣
 ↪as suggested by neuralNer

crf = CRF(n_tags)  # CRF layer
out = crf(model)  # output

model = Model(model_input, out)
model.compile(optimizer="rmsprop", loss=crf.loss_function, metrics=[crf.
 ↪accuracy])
model.summary()

history = model.fit(X_tr,
                    np.array(y_tr),
                    batch_size=32,
                    epochs=5,
                    validation_split=0.1,
                    verbose=1)
hist = pd.DataFrame(history.history)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 75)                0
_________________________________________________________________
embedding_3 (Embedding)      (None, 75, 20)            703580
_________________________________________________________________
bidirectional_3 (Bidirection (None, 75, 100)           28400
_________________________________________________________________
time_distributed_3 (TimeDist (None, 75, 50)            5050
_________________________________________________________________
crf_3 (CRF)                  (None, 75, 17)            1190
=================================================================
Total params: 738,220
Trainable params: 738,220
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /Users/aditya/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:986: The name tf.assign_add is
deprecated. Please use tf.compat.v1.assign_add instead.

Train on 38846 samples, validate on 4317 samples
Epoch 1/5
38846/38846 [==============================] - 523s 13ms/step - loss: 0.1129 -
crf_viterbi_accuracy: 0.9685 - val_loss: 0.0424 - val_crf_viterbi_accuracy:
```

```
0.9852
Epoch 2/5
38846/38846 [==============================] - 547s 14ms/step - loss: 0.0329 -
crf_viterbi_accuracy: 0.9878 - val_loss: 0.0272 - val_crf_viterbi_accuracy:
0.9895
Epoch 3/5
38846/38846 [==============================] - 516s 13ms/step - loss: 0.0241 -
crf_viterbi_accuracy: 0.9903 - val_loss: 0.0236 - val_crf_viterbi_accuracy:
0.9902
Epoch 4/5
38846/38846 [==============================] - 574s 15ms/step - loss: 0.0209 -
crf_viterbi_accuracy: 0.9913 - val_loss: 0.0223 - val_crf_viterbi_accuracy:
0.9906
Epoch 5/5
38846/38846 [==============================] - 561s 14ms/step - loss: 0.0191 -
crf_viterbi_accuracy: 0.9919 - val_loss: 0.0219 - val_crf_viterbi_accuracy:
0.9906
```

[8]:
```
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.figure(figsize=(12,12))
plt.plot(hist["crf_viterbi_accuracy"])
plt.plot(hist["val_crf_viterbi_accuracy"])
plt.show()
```



[11]:
```
# Evaluation
from seqeval.metrics import precision_score, recall_score, f1_score,␣
 ↪classification_report

test_pred = model.predict(X_te, verbose=1)
d_index_to_tags = {i: w for w, i in d_indexed_tags.items()}

def pred2label(pred):
    out = []
    for pred_i in pred:
        out_i = []
        for p in pred_i:
            p_i = np.argmax(p)
            out_i.append(d_index_to_tags[p_i].replace("PAD", "O"))
        out.append(out_i)
```

```
    return out

pred_labels = pred2label(test_pred)
test_labels = pred2label(y_te)
print("F1-score: {:.1%}".format(f1_score(test_labels, pred_labels)))
print(classification_report(test_labels, pred_labels))
```

```
4796/4796 [==============================] - 15s 3ms/step
F1-score: 82.9%
           precision    recall  f1-score   support

      tim       0.89      0.83      0.86      2046
      geo       0.83      0.90      0.87      3723
      per       0.78      0.73      0.76      1738
      org       0.72      0.68      0.70      2017
      gpe       0.97      0.94      0.96      1629
      nat       0.00      0.00      0.00        18
      art       0.00      0.00      0.00        41
      eve       0.83      0.17      0.29        29

micro avg       0.84      0.82      0.83     11241
macro avg       0.83      0.82      0.83     11241
```

[18]:
```
# Trying some predictions
idx = 5
p = model.predict(np.array([X_te[idx]]))
p = np.argmax(p, axis=-1)
true = np.argmax(y_te[idx], -1)
print("{:15}||{:5}||{}".format("Word", "True", "Pred"))
print(30 * "=")
for w, t, pred in zip(X_te[idx], true, p[0]):
    if w != 0:
        print("{:15}: {:5} {}".format(l_words[w-1], l_tags[t], l_tags[pred]))
```

```
Word           ||True ||Pred
==============================
The            : O     O
statement      : O     O
says           : O     O
five           : O     O
other          : O     O
al-Qaida       : B-org B-org
members        : O     O
suspected      : O     O
of             : O     O
facilitating   : O     O
```

```
roadside       : O     O
bombings       : O     O
in             : O     O
Taji           : B-geo B-geo
Qada           : I-geo I-geo
,              : O     O
northwest      : O     O
of             : O     O
Baghdad        : B-geo B-geo
,              : O     O
were           : O     O
also           : O     O
detained       : O     O
.              : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
Gianfranco     : O     O
```

```
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
Gianfranco      : O     O
```

[23]:
```
# Prediction on new sentence
test_sentence = ["Hawking", "was", "a", "Fellow", "of", "the", "Royal",␣
 ↪"Society", ",", "a", "lifetime", "member",
                  "of", "the", "Pontifical", "Academy", "of", "Sciences", ",",␣
 ↪"and", "a", "recipient", "of",
                  "the", "Presidential", "Medal", "of", "Freedom", ",", "the",␣
 ↪"highest", "civilian", "award",
                  "in", "the", "United", "States", "."]
x_test_sent = pad_sequences(sequences=[[d_indexed_words.get(w, 0) for w in␣
 ↪test_sentence]],
                            padding="post", value=0, maxlen=n_max_seq_size)
p = model.predict(np.array([x_test_sent[0]]))
p = np.argmax(p, axis=-1)
print("{:15}||{}".format("Word", "Prediction"))
print(30 * "=")
for w, pred in zip(test_sentence, p[0]):
    print("{:15}: {:5}".format(w, l_tags[pred]))
```

```
Word           ||Prediction
==============================
Hawking        : B-tim
was            : O
a              : O
Fellow         : O
of             : O
the            : O
Royal          : B-org
Society        : I-org
,              : O
```

```
a               : O
lifetime        : O
member          : O
of              : O
the             : O
Pontifical      : B-org
Academy         : I-org
of              : I-org
Sciences        : I-org
,               : O
and             : O
a               : O
recipient       : O
of              : O
the             : O
Presidential    : O
Medal           : B-tim
of              : O
Freedom         : B-geo
,               : O
the             : O
highest         : O
civilian        : O
award           : O
in              : O
the             : O
United          : B-geo
States          : I-geo
.               : O
```

[ ]: